Déploiement logiciel open source et multiplateforme

Guillaume Ferry
Université de Lorraine
Direction du numérique - Sous-Direction Service aux Usagers
Campus Aiguillettes
54500 Vandoeuvre-les-Nancy

Résumé

Qui n'a jamais rêvé d'une solution unique pour se débarrasser des tâches d'administration logicielles sous Linux, Windows et macOS?

L'évolution du champ d'action et des domaines de compétences nous pousse à rationaliser les tâches les moins valorisantes et les plus chronophages de notre travail ; la gestion du cycle de vie logiciel d'un parc en fait partie. Installation du poste de travail, déploiements ponctuels, mises à jour d'applications, sont autant de tâches manuelles répétitives que nous avons cherché à automatiser et à fiabiliser.

Nous avons conclu à l'issue d'une étude préalable qu'aucune solution clé en main ne pourrait répondre entièrement au contexte de l'université :

- *Disposer d'une solution unique, multi-site, multiplateforme et open source ;*
- *Disposer d'une interface simple, intuitive et accessible depuis n'importe où ;*
- *Offrir un catalogue logiciel assez étendu pour répondre aux besoins d'une population d'utilisateurs variés ;*

C'est en associant plusieurs « briques » logicielles indépendantes que nous avons pu répondre à cette problématique. Ce modèle est bâti autour d'un service de gestion de configuration centralisé capable de communiquer avec la majorité des gestionnaires de paquets. L'exploitation de cette solution se fait au travers d'une application web qui a fait l'objet d'un développement interne.

Mots-clefs

PUPPET, CHOCOLATEY, NEXUS, DEVOPS, DEPLOIEMENT LOGICIEL

1 Introduction

1.1 Contexte

Le campus de la faculté des sciences de l'université de Lorraine compte 4700 étudiants et 560 personnels enseignants-chercheurs, administratifs et techniques. Le parc informatique qui est composé d'environ 2200 machines, s'étend sur 100000m2 de locaux regroupant entre autres 15 unités de recherche. La répartition des systèmes d'exploitation est de l'ordre de 1600 machines *Windows*, 250 *Mac* et 350 machines *Linux*.

1.2 Problématique

Le périmètre à couvrir, la diversité des systèmes et la mixité des populations d'utilisateurs nous ont conduit en 2015 à mener une réflexion sur l'automatisation du déploiement de logiciels. Installation du poste de travail, déploiements ponctuels, mises à jour d'applications, sont autant de tâches manuelles répétées et source d'erreurs que nous avons cherché à automatiser et à fiabiliser.

Les solutions de déploiements logiciels du moment ne permettaient pas de répondre à l'ensemble des besoins que nous avions identifiés :

- Disposer d'une solution unique, multiplateforme et open source ;
- Disposer d'une interface simple et intuitive accessible depuis n'importe où ;
- Disposer d'un catalogue logiciel étendu et évolutif, en adéquation avec les logiciels utilisés sur le campus ;
- Pouvoir gérer des profils logiciels pour des populations d'utilisateurs ;
- Garantir une homogénéité des logiciels déployés sur le parc ;
- Maintenir le cycle de vie logiciel des machines ;
- Disposer d'une solution utilisable pour des déploiements ponctuels ou en post-installation.

2 Solutions

Nous avons choisi de répondre à ces problématiques en pilotant les gestionnaires de paquets des machines clientes depuis un outil de gestion de configuration centralisé. Déjà présents sur la majorité des distributions Linux et désormais sous Windows 10, les gestionnaires de paquets permettent de manipuler silencieusement les tâches d'installation, de désinstallation et de mises à jour. Ils garantissent également l'authenticité des sources d'installation et le contrôle des versions disponibles sur le(s) dépôt(s) logiciel(s) dont ils dépendent.

Les gestionnaires de configurations sont des outils permettant d'agir à distance sur des ressources locales, telles que l'état des services système, les fonctions du gestionnaire de paquets ou encore le système de fichiers.

2.1 Gestion de configuration centralisée avec *Puppet*

Puppet est un outil de gestion de configuration centralisée open source et multiplateforme de type *client – serveur.* Son langage de programmation dédié *(Domain Specific Language)* permet de décrire un état souhaité et de l'appliquer, à l'aide d'un agent, sur des ordinateurs distants.

Réservé dans un premier temps à la configuration de nos serveurs, *Puppet* est devenu en quelques années un outil incontournable pour administrer l'ensemble des systèmes Linux du campus. Au fil du temps, la perspective d'une utilisation sur d'autres plateformes s'est dessinée.

2.1.1 Communiquer de manière sécurisée

Pour sécuriser les échanges avec les agents, le serveur *Puppet* embarque des fonctionnalités de *PKI* et *d'autorité de certifications* (*CA*). Une fois autorisés par le serveur au travers d'une procédure de signature de certificat, les agents peuvent initier une communication qui se déroule en deux étapes :

- Obtenir les instructions à exécuter auprès du serveur *(Catalogue)* ;
- Transmettre en retour un rapport détaillé sur l'exécution du *catalogue* ainsi qu'une collection d'informations système appelées *facts* ;

L'agent s'exécute en tant que service et initie une communication avec le serveur à intervalles réguliers.

2.1.2 Définir et maintenir un état souhaité

Les instructions permettant d'atteindre un état précis sont rédigées dans le langage déclaratif propre à *Puppet (DSL)* qui présente de nombreux avantages en comparaison des scripts traditionnels :

- **Indépendance du système d'exploitation cible**

L'état souhaité des composants systèmes (Utilisateurs et groupes, systèmes de fichiers, packages, services, etc …) est décrit de manière abstraite dans des fichiers d'extensions *.pp*, au travers d'objets appelés *Resources*.

Cet exemple illustre l'instanciation d'une *ressource* « *Package »* permettant de maintenir Firefox installé dans sa dernière version :

```
class firefox {

  package {'firefox':
       ensure => latest,
  }
}
```

Derrière ces *ressources* à la sémantique généraliste se cachent des fonctions *(providers),* chargées de traduire l'état décrit, en commandes propres à l'environnement de l'agent. Pour la ressource *package* par exemple, on distinguera autant de *providers* que de gestionnaires de paquets supportés. Le contexte étant différent d'un système à un autre (*S*ystème d'exploitation*, facts,* variables*,* résultats de fonctions*, etc* …), les instructions compilées dans le catalogues sont variables sur la forme.

- **Un langage extensible porté par une forte communauté**

De nombreuses *ressources* complémentaires sont disponibles sur la forge Puppetlabs. (*Powershell*, *Chocolatey*, *homebrew*, Registre Windows, Services *apache*, *zabbix*, *postfix*, *nginx*…) Il est également possible de développer ses propres *ressources*.

- **Une structuration du code modulaire**

Les *modules* permettent d'associer plusieurs *ressources* entre elles et d'exprimer leurs dépendances.

Dans cet exemple, le module *matlab2019a* décrit le processus d'installation d'un paquet nécessitant un fichier de licence. La ressource *package* permettra de s'assurer que le paquet est installé tandis que la ressource *file* garantira la présence du bon fichier de licence au bon endroit. La notion de dépendance entre les 2 ressources est exprimée avec l'attribut *require* (La réalisation de *File* nécessite la réalisation de *Package*).

```
class matlab2019a {

  package { 'matlab2019a':
    ensure => present,
  }

  file { '/etc/matlab/licence.lic' :
    ensure  => present,
    source  => 'puppet:///modules/matlab/licence.lic',
    require => Package['matlab2019a'],
  }
}
```

L'association entre *modules* et *agents (nodes)* s'écrit dans des *manifests*.

L'exemple suivant illustre un *manifest (foo.pp)* dans lequel le *module matlab2019a* est attribué à la machine *foo.jres.org*.

```
node 'foo.jres.org' {
  include matlab2019a
}
```

Les *manifests* sont classés dans des *environnements*. Outre leur vocation organisationnelle, les *environnements* sont paramétrables et permettent de définir des instructions pour l'ensemble des *manifests* qu'il contiennent.

Lors de chaque exécution *(run)*, l'agent *Puppet* télécharge et exécute ses instructions compilées dans le *catalogue* qui lui est propre.

2.2 Le gestionnaire de paquets *Chocolatey*

2.2.1 Introduction

Depuis 2015, nous avons volontairement axé nos travaux sur l'environnement Windows compte tenu de sa répartition majoritaire sur le parc. Nous avons choisi d'associer les fonctionnalités d'automatisation de *Puppet* à *Chocolatey* qui est un gestionnaire de paquets open source pour *Windows.* En effet, même si *Windows 10* peut supporter plusieurs gestionnaires de paquets simultanément à l'aide de son agrégateur *OneGet*, il n'en existe pas d'officiel. *Chocolatey* repose sur *NuGet*, le gestionnaire de paquets *Microsoft* dédié aux développements de composants *.NET framework*. Un ensemble de fonctions *Powershell (Helpers)* forment une couche d'abstraction supplémentaire permettant de s'approprier la philosophie des gestionnaires de paquets Linux traditionnels.

Tout comme *YUM* et *APT*, les paquets sont hébergés sur des dépôts qui peuvent être publics ou privés. Une communauté active dont nous faisons partie étoffe et maintien le catalogue logiciel à jour. En constante évolution, ce catalogue compte plus de 7000 paquets en téléchargement public en octobre 2019. Les paquets les plus populaires tels que Adobe Reader (~ 160 000 000 de téléchargements), Google Chrome (~ 20 000 000 de téléchargements), ou encore Firefox (~ 10 000 000 de téléchargements) sont packagés quelques heures après la sortie d'une nouvelle version chez son éditeur.

La complémentarité entre *Chocolatey* et *Puppet* a fortement influencé notre choix. En effet dès 2015, le support du gestionnaire de paquets dans *Puppet* a été officiellement pris en charge par *Puppetlabs* au travers d'une *ressource* complémentaire. Au sens de la *ressource package, Chocolatey* est devenu un *provider* au même titre que *YUM* ou *APT*.

Exemple d'instanciation de la ressource Package permettant d'installer le paquet Firefox avec *Chocolatey*.

```
class firefox {

  package {'firefox':
       ensure => latest,
       provider => chocolatey,
  }
}
```

2.2.2 Utilisation

Lorsque *Chocolatey* est installé sur un système, la commande *choco* permet de gérer les paquets en ligne de commande comme sur une distribution Linux.

- Installation du paquet firefox

```
PS C:\> choco install firefox
```

- Mise à jour du paquet firefox dans sa dernière version disponible

```
PS C:\> choco upgrade firefox
```

- Lister les paquets installés localement

```
PS C:\> choco list -lo
```

- Désinstallation d'un paquet

```
PS C:\> choco uninstall firefox
```

- Lister les dépôts configurés localement

```
PS C:\> choco source
```

2.2.3 Gestion de paquets

Un paquet *Chocolatey* est une archive portant l'extension *.nupkg.* On y retrouve au minimum :

- Un fichier de métadonnées d'extension *.nuspec* contenant des informations sur le paquet (Auteur, version, description, etc.) ;
- Un script *powhershell* nommé *chocolateyinstall.ps1* contenant les instructions d'installation du logiciel packagé ;

Extrait du fichier de métadonnées *algobox.nuspec* contenu dans le paquet public algobox

```
<?xml version="1.0" encoding="utf-8"?>
<package xmlns="http://schemas.microsoft.com/packaging/2015/06/nuspec.xsd">
  <metadata>
    <id>algobox</id>
    <version>1.0.2</version>
   <title>algobox (Install)</title>
  <authors>Pascal Brachet - Lycée Bernard Palissy d'Agen</authors>
  <owners>Guillaume Ferry - DN-SU Université de Lorraine</owners>
  <projectUrl>http://www.xm1math.net/algobox/index.html</projectUrl>
  <packageSourceUrl>https://github.com/guillaume-ferry/algobox/tree/master</
packageSourceUrl>
    <tags>algobox</tags>
    <summary>Initiation à l'algorithmique</summary>
  <docsUrl>http://www.xm1math.net/algobox/doc.html</docsUrl>
  <description>
  AlgoBox is an opensource, cross-platform, and free software for developing and executin
g algorithms in the spirit of new high school math programs.
  AlgoBox est un logiciel libre, multi-plateforme et gratuit d'aide à l'élaboration et à
l'exécution d'algorithmes dans l'esprit des nouveaux programmes de mathématiques du seco
ndaire.
  </description>
  </metadata>
  <files>
    <file src="tools\**" target="tools" />
  </files>
</package>
```

- Extrait du fichier *chocolateyinstall.ps1* du paquet public algobox

```
$ErrorActionPreference = 'Stop';
$toolsDir   = "$(Split-Path -parent $MyInvocation.MyCommand.Definition)"
$url64      = 'http://www.xm1math.net/algobox/assets/files/Algobox_1.0.2_Win_x64.msi'

$packageArgs = @{
      packageName   = $env:ChocolateyPackageName
      unzipLocation = $toolsDir
      fileType      = 'MSI'
      url64bit      = $url64
      softwareName  = 'algobox*'
      checksum64    = '4C5A3975AD8916ADD9887C62FB617375S3477FA939A537B37BA809SA916DE38'
      checksumType64= 'sha256'

      silentArgs    = "/qn /norestart /l*v `"$($env:TEMP)\$($packageName).$
($env:chocolateyPackageVersion).MsiInstall.log`""
      validExitCodes= @(0, 3010, 1641)
}

Install-ChocolateyPackage @packageArgs
```

A l'aide des informations définies dans *chocolateyinstall.ps1*, le *helper Install-ChocolateyPackage* est en mesure de réaliser :

1. Le téléchargement et la vérification des sources ;
2. L'installation ou la mise à jour silencieuse du logiciel en tenant compte de l'architecture du système et du type d'installeur (*MSI*, *installShield*, etc.) ;
3. La gestion des erreurs et des logs.

Selon les conditions de redistributions définies par la licence, les sources peuvent être intégrées dans le paquet, ou localisées au travers d'un lien de téléchargement vers le site de l'éditeur.

2.2.4 Sécurité

La simplicité du code et sa transparence sont un gage de confiance mais ne suffisent pas à garantir la sécurité du processus d'installation d'un paquet. C'est la raison pour laquelle chaque paquet public est soumis à une chaine de validation avant d'être publié. Elle se veut dans un premier temps automatisée (Vérifications sémantique, contrôle antivirus, contrôles d'intégrité), puis humaine (Modération).

Les échanges entre agents et dépôts sont entièrement chiffrés *(https),* tandis que l'intégrité des paquets est systématiquement contrôlée (*emprunte SHA512*). Des mesures de sécurité supplémentaires, telles que la signature cryptographique des paquets sont annoncées dans les prochaines versions de *Chocolatey*.

La création de paquets est accessible et sa courbe d'apprentissage est rapide, notamment grâce à une documentation de qualité. La maitrise de *Powershell* reste cependant un prérequis pour envisager des scénarios d'installation complexes.

2.3 Packaging, hébergement et diffusion de paquets

La majorité des logiciels métiers utilisés dans notre périmètre sont soumis à des licences interdisant leur redistribution. Le développement et l'hébergement de nos propres paquets sont de nouvelles problématiques (Maitrise du périmètre de distribution - respect des licences - persistance des sources – qualité / validation / organisation du code, etc.) que nous avons solutionnées en deux étapes.

2.3.1 Hébergement de paquets avec *Nexus Repository*

Nexus Repository est un logiciel open source destiné au stockage et la distribution de composants logiciels. Il supporte de nombreux formats tels que *Apt, Yum, Nuget, Maven,* ou encore *RAW* et s'intègre de ce fait dans des environnements multiplateformes. Nous l'utilisons pour héberger et distribuer les paquets développés en interne. En parallèle, nous implémentons une fonctionnalité de *proxy/cache* propre à Nexus qui permet de diminuer les flux réseaux sortants vers les dépôts officiels.

- L'exemple suivant illustre la configuration d'un dépôt *Nexus* avec *Chocolatey :*

```
PS C:\> choco source add -n=my_repo `
-s="https://myrepo.local/repository/chocolatey"`
--priority=1
```

- L'exemple suivant illustre l'équivalent sous Ubuntu :

```
#/etc/apt/source.list.d/myrepo.list
deb https://myrepo.local/repository/ubuntu bionic main
```

L'exemple suivant illustre le *module puppet myrepo* dans lequel la *ressource chocolateysource* est décrite :

```
class myrepo {

  chocolateysource {'myrepo':
    ensure    => present,
    location  => 'https://myrepo.local/repository/chocolatey/',
    priority  => 1,
  }

}
```

2.3.2 Développement et intégration continue avec GITLAB

Pour répondre à une demande grandissante de création de paquets, nous avons organisé nos développements autour d'une forge *gitlab* que nous hébergeons et maintenons dans sa version open source. De nombreuses raisons nous ont amené à utiliser ce type d'outils notamment :

- L'aspect collaboratif permettant entre autres d'uniformiser nos pratiques ;
- Le suivi de version de code avec *git ;*
- Les fonctionnalités *CI/CD* permettant d'automatiser le contrôle et la diffusion du code.

Le processus de développement et d'intégration continue (*CI/CD*) est un mécanisme employant un ou plusieurs systèmes esclaves (*runner*) chargés d'exécuter un ensemble de tâches (*Jobs)* visant à tester, valider et déployer le code fraîchement mis à jour sur la forge.

Chocolatey fourni son propre environnement de test, permettant de valider la syntaxe et d'interpréter les étapes d'installation et de désinstallation d'un paquet. Nous avons intégré cet environnement dans notre processus *CI/CD.*

un rapport détaillé sur l'exécution du *job* est retourné au serveur *Gitlab* et transmis par mail aux contributeurs du projet.

2.4 Interface d'administration

L'association de ces 3 outils permet de répondre de manière pertinente aux problématiques techniques identifiées.

Cependant, l'exploitation de la solution en l'état se limiterait à un groupe restreint d'initiés. Pour rendre cet outil simple et accessible, nous avons développé un portail web d'administration dont le niveau d'abstraction est suffisamment élevé pour permettre à n'importe quel gestionnaire de parc informatique d'être opérationnel en quelques minutes.

Sans pour autant négliger le champ d'action des ressources de *Puppet,* nous avons volontairement focalisé notre développement sur la problématique du déploiement logiciel. Voici une liste non exhaustive des fonctionnalités de l'interface :

- Authentification CAS ;
- Gestion de profils logiciels ;
- Gestion des machines / groupes de machines ;
- Délégation d'administration multi-site ;
- Tableau de bord.

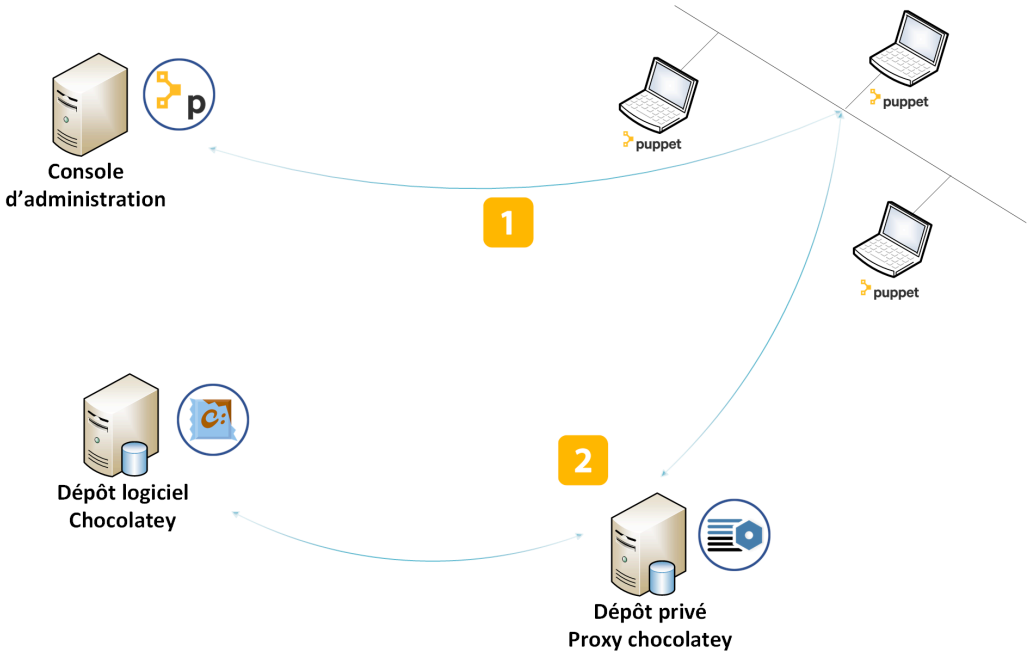
Les bénéfices de l'interface sont multiples. Elle permet de consulter et modifier d'une manière simple et synthétique les informations codées dans *Puppet*. (Affectation de profils logiciels, gestion des groupes, conformité des machines, *Facts puppet*, etc.).

Elle garantit la cohérence du code et organise son cycle de vie en automatisant la gestion de version du code généré.

Enfin elle rend autonome les services de proximité sur l'administration de leurs sites en s'appuyant sur les *environnements Puppet*.

3 Infrastructure

3.1 Schéma de l'infrastructure



1 Les agents recueillent leurs stratégies sur le serveur

2 Les logiciels à installer sont téléchargés depuis le dépôt/proxy privé.

3.2 Configuration matérielle

L'ensemble des serveurs sont virtualisés sur l'infrastructure virtuelle de l'université de Lorraine.

Service Puppetmaster

- Serveur PUPPET : 8 coeurs / 16Go RAM / 200Go HDD
- Serveur POSTGRES (puppetDB) : 8 coeurs / 16Go RAM / 200Go HDD

Service GITLAB

- Serveur GITLAB: 2 coeurs / 16 Go RAM / 100Go HDD
- Runner : 2 coeurs / 16 Go RAM / 100Go HDD

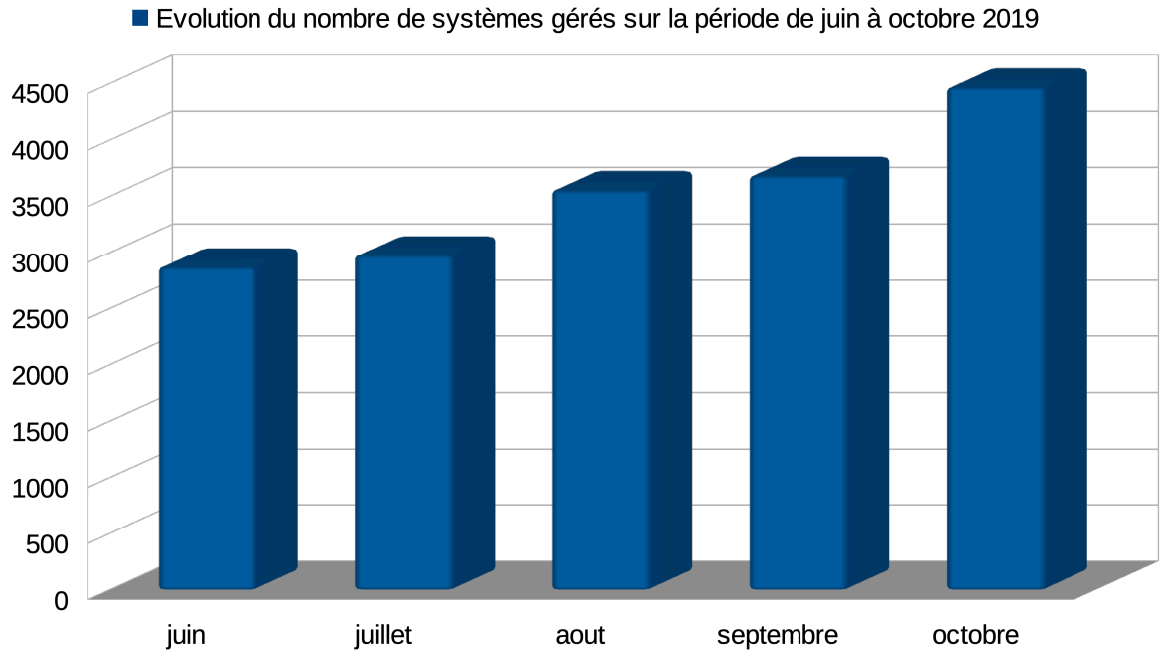
Service NEXUS

- dépot1 :VM – 16 coeurs / 16Go RAM / 300 Go HDD
- dépot2 :VM – 16 coeurs / 16Go RAM / 300 Go HDD

4 Conclusion

4.1 Vers une solution à l'échelle de l'université

Nous avons présenté ce service aux autres équipes informatiques de proximité de l'Université de Lorraine, à l'occasion d'un séminaire portant sur les solutions de déploiements. Nombre d'entre elles ont exprimé le souhait de pouvoir la tester. Durant une phase d'expérimentation qui a débuté en 2018, nous avons obtenu différents retours qui nous ont permis d'améliorer certains aspects. Suite à cela, cette solution qui était en production sur le campus de la faculté des Sciences de Nancy, a été retenue par la direction du numérique comme l'une des options possibles pour répondre à la problématique du déploiement logiciel à l'échelle de l'université. La migration des serveurs vers l'infrastructure de l'université touche à sa fin. Actuellement, plus de la moitié des équipes de site utilisent la solution en production, avec à ce jour 4500 postes de travail déclarés.



4.2 Bilan

Par l'association de différentes briques logicielles open source, nous avons pu répondre de manière globale aux problématiques soulevées en 2015 et en tirer les bénéfices sur le terrain.

En effet, les retours des équipes de site mettent en avant un réel gain de temps au quotidien en comparaison à l'investissement nécessaire pour s'approprier la solution. Les opérations d'installation qui pouvaient durer plusieurs jours se mesurent aujourd'hui en minutes.

Modulaire, la solution est également évolutive, grâce à la notion de *providers*. L'implémentation récente du gestionnaire de paquets *homebrew* pour *macOS* en est l'exemple. Ce mode de fonctionnement nous permet d'envisager plus sereinement les évolutions liées aux systèmes d'exploitation sans remettre en cause l'ensemble de la solution.

4.3 Perspectives

Le parc informatique de l'université (16000 machines) et son périmètre (une cinquantaine de sites répartis sur 3 départements) fixent de nouveaux enjeux pour le groupe de travail.

L'évolution constante des accès aux sources logicielles requiert un investissement supplémentaire pour fiabiliser les accès (haute disponibilité / répartition de charge) et sécuriser les méthodes d'accès aux dépôts (*PKI*).

Les résultats satisfaisants issus de la phase d'expérimentation du déploiement logiciel sous *macOS* nécessitent de nouvelles actions pour atteindre une qualité de service équivalente à celle des environnements *Linux* et *Windows*.

En complément, la diversité des *ressources Puppet* ouvre une piste de réflexion sur la personnalisation de l'environnement utilisateur et la configuration système.